https://doi.org/10.31449/inf.v46i4.3565 Informatica **46** (2022) 507–522 **507**

Parametrized MTree Clusterer for Weka

Marian Cristian Mihăescu, Marius Andrei Ciurez
Department of Computer Science and Information Technologies, Faculty of Automatics, Computers and Electronics, University of Craiova, Craiova, Romania
E-mail: cristian.mihaescu@edu.ucv.ro, mariusandrei.ciurez@gmail.com

Keywords: Clustering, MTrees, metric spaces

Received: May 25, 2021

In the area of clustering, proposing or improving new algorithms represents a challenging task due to an already existing well-established list of algorithms and various implementations that allow rapid evaluation against tasks on publicly available datasets. In this work, we present an improved version of the MTree clustering algorithm implemented within Weka workbench. The algorithmic approach starts from classical metric spaces and integrates parametrised business logic for finding the optimal number of clusters, choosing the division policy and other characteristics. The result is a versatile data structure that may be used in clustering to find the optimal number of clusters, but mainly for loading data sets that already have a known structure. Experimental results show that MTree finds the proper structure in two clustering tasks, although other algorithms fail in various ways. A discussion of further improvements and experiments on real data sets and functions is included.

Povzetek: Opisana je nova metoda algoritma MTree za gručenje.

1 Introduction

As an unsupervised learning technique, clustering is continuously getting attention within the machine learning area due to many available algorithms and a wide range of application domains for which practical solutions are continually being designed and implemented.

The general problem of building clusters of objects is narrowed down in [41] by clearly stating that the first thing that needs to be taken into consideration in the context of the problem. Therefore, building a general-purpose clustering algorithm that may work with any objects and solve any task is not a realistic option. Thus, the objects we need to define and provide a proper task description accompanied by particular distance and evaluation metrics or various algorithmic approaches need to be carefully taken care of in building a clustering data analysis workflows.

In [1], authors recently raised the problem of clusterability of a dataset. Having an available dataset does not necessarily imply that we may meaningfully run a clustering process to solve a particular task. Thus, defining clusterability becomes a critical issue. Checking if a dataset is clusterable becomes an inherently tricky problem, especially when dealing with actual data and solving a particular practical task.

The wide range of approaches used in available implemented clustering algorithms has found various application domains to provide efficient solutions to many practical tackled problems. From the many application domains, we mention medical image processing (i.e., pattern recognition and image segmentation) [39] [24], general and natural language processing knowledge discovery [20] [33] [34], navigation of robots [14] [22] and in many other contexts.

In the area of unsupervised learning, there are several general classes of clustering algorithms (i.e., flat, hierarchical and density-based) that all share two common problems: finding the optimal number of clusters and quickly and efficiently finding the correct clusters taking into consideration specific distance measures appropriate for the objects (i.e., pixels, points, persons, books, etc.) that are being grouped. Further, once the clusters are being correctly determined, there may be later used to query for the nearest neighbours or run specific range queries. Depending upon the inner data structures used for managing the clusters when tree data structures are being used efficiently, searching or traversing may be accomplished efficiently

The objective of this work is to present an improved version of the metric trees (MTree) algorithm that has been firstly proposed by [7] and later by [40] and [43].The first proposal of using MTrees in the context of clustering has been made in [31] and later implemented as Weka [16] package in [30]. This paper presents an improved version of the works from [30] and [31] that has been tested in a comparative benchmark with k-MS morphological reconstruction clustering algorithm [37] along with classical algorithms (i.e., simple k-means, Cobweb, Farther First and Canopy) in [9].

This paper presents a further improved sparametrised version of the MTree algorithm in terms of managed items, used distance, the method for finding and setting K (i.e., the number of clusters), division policy and validation metrics. The critical improvement of the new parametrised version of the MTree clusterer is that it is now suitable to address a broader range of problems. The parametrised version is

not ideal for solving any clustering problem by dynamically choosing the suitable parameters. Instead, a wide range of clustering problems may be addressed to the newly proposed MTree cluster by properly setting its parameters depending on the available data objects and the tackled task.

In general, there are two types of clustering problems. One regards finding patterns in a dataset that we do not know if they have a specific structure or if a certain number of clusters exist. The second type of task regards correctly building a data clustering model that may later be queried many times for getting specific information about managed data. In this scenario, the cluster model is constructed only once such that it may be regarded as a preprocessing step. Later, few insertions and updates may occur at runtime, while most calls are range queries or kNN queries that need to be solved correctly and efficiently. The proposed approach is suited for both tasks, but the second one is more appropriate. Range queries determine items whose distance from a specific query item is smaller than a particular value.

The paper is organised as follows. In Section 2, we perform a literature review with regards to similar libraries other application domain usages of metric trees for clustering such as time series analysis of cytometry data, recommender systems, spatial clustering data, automatic computing the number of clusters for colour or greyscale image segmentation and clustering quality metrics. Section 3 describes the proposed approach with a detailed presentation of how each parameter may be set and how it influences the business logic of the MTree. Section 4 presents the experimental results on two publicly available data sets with runs on several parameter settings. Finally, section 5 contains the conclusions of this work, summarises the main contributions and discusses potential improvements and applications.

2 Related work, limitations and approaches

Data clustering (also known as unsupervised learning) represents a subarea of machine learning. Other areas of machine learning are supervised learning, reinforcement learning or deep learning. A particular sub-domain is represented by age clustering and segmentation [13] which presents the use of subtractive clustering along with classical K-means algorithm to preprocess the data for optimal centroid initialisation. The experimental results were obtained on medical images representing infected blood cells with malaria. The classical images used for segmentation bring better results than k-means taking into account root mean square error (RMSE) and peak signal-to-noise ratio (PSNR) metrics.

Another approach for image clustering was proposed by Chang et al. in [5]. They propose a Deep Adaptive Clustering (DAC) approach that reduces to a classification problem in which similarity is determined by cosine distance and learned labelled features tend to be one-hot vectors obtaining good results on popular and accessible datasets like MNIST, CIFAR-10 and STL-10.

One of the critical practical usages of clustering algorithms is for image segmentation. Including spatial information along with taking outlier points at a later stage in the clustering algorithm has been proposed in [42]. From this perspective, outliers are data points with almost equal distance to their adjacent clusters and therefore should be taken into consideration later. This approach raises two issues. One regards the fact that the order in which data points are given to the algorithm has significant importance. Thus, if a clustering algorithm is highly sensitive to the order in which data points are provided with a custom preprocessing may be necessary. The second issue regards the very nature of the data set from the clusterability perspective. The critical verification that is also highly recommended is to always check for clusterability before starting to the clustering analysis.

Another application of clustering is grey scale image segmentation [44]. As compared with the clustering of colour images or with images that incorporate spatial information, the task is to highly decrease the time complexity of the algorithm by using affinity propagation (AP) clustering algorithm. As always, when real life grey scale images are being clustered the problem of correctly determining the number of clusters or segments is a critical one.

More elaborate applications regard indexing and retrieval of similar images from an image database (CBIR - Content-Based Image Retrieval) which represent a challenging task that has been addressed in [29] and [27]. The first approach uses features, colour and texture. It employs K-means and hierarchical clustering for finding the most similar images. The second approach uses colour, texture and shape as features and K-means as business logic for determining four classes of images: dinosaurs, flowers, buses and elephants. The obtained experimental results are promising in terms of excellent precision and recall values.

Clustering has also been used successfully in recommender systems that were based on collaborative filtering in [11]. A novel K-medoids clustering recommendation algorithm has been proposed, which introduced an improved Kullback-Leibler divergence for computing item similarity. The final task was to improve the effectiveness of the developed recommendation system.

Lately, in the application domain of immunology has been used clustering algorithms - *ChronoClust*, a new density-based clustering algorithm - on time series cytometry data [26]. The task was to characterise the immune response to disease by tracking temporal evolution.

Very recent works [1] put a high emphasis on defining clusterability and checking if a dataset is clusterable as a preprocessing step before any other data analysis is further performed. Therefore, before applying the algorithm for solving a task that requires clustering a *sanity check* may be required, in the way that we should verify the clusterability of the dataset. In other words, clustering may not work

on datasets which do not exhibit any internal structure, irrespective of any particular algorithm that may be employed. In our case, clusterability becomes a prerequisite that the dataset needs to meet before being loaded into the MTree structure. The new proposed clustering algorithm should have as main scenario working with data that is known to be clusterable, for which we know it has a well-defined structure with a known number of clusters. A dataset has a well-defined structure when there exists an assignment of items into clusters that is validated by a domain specialist for real world datasets. In the case of synthetic datasets, the function that creates the instances is designed in such a way that clusters are well-defined and represent the gold labeling for any clustering algorithm.

If the number of clusters is not known than the results highly depend on the particular practical context of the clustering problem. The context is defined by the problem (clustered objects and clustering task) and parameters of the MTree: object type, distance function, splitting policy and validation metrics. If the dataset is not clusterable, we argue that the MTree algorithm - as well as any other clustering algorithms - will exhibit undefined behaviour.

A more complex context occurs when the image source is unknown or when the ground truth for the training dataset is also unknown [4] [10]. In this particular situation, finding the optimal K represents am an inherently difficult task. From an experimental algorithmic perspective, using proper distance metrics and loss function in this optimisation problem represents one of the key ingredients towards successful results. Such approaches propose fancy solutions such as hierarchical clustering or clustering ensembles based graph partitioning methods, cluster-based Similarity Partitioning Algorithm (CSPA), HyperGraph Partitioning Algorithm (HGPA), and Meta Clustering Algorithm (MCLA).

Among the most well-known issues in unsupervised learning consists in determining the actual number of clusters from a dataset. Unfortunately, scenarios in which the value of K is known to occur only in a few practical systems. In general, an application that performs an image processing task does not have any information regarding the actual number of clusters from the target image. This situation may occur when dealing with data streams [25] or with very high-dimensional datasets [17]. In general, one of the most suitable approaches tries to reduce to automatic determination of K that may be based on dynamic clustering [17] or joint tracking segmentation [32].

A general-purpose algorithm for finding the optimal number of clusters has been proposed by [6] and implemented in an R package in NbClust. The main idea of this approach is that it may use up to 30 indices for voting the number of clusters. The package has implemented a function to run a clustering algorithm (i.e., k-means or hierarchical clustering) using various distance measures and aggregation methods. The main limitation of the approach is that it is general and practical usage for particular datasets needs to be parametrised by the appropriate clustering algorithm, subset of indices, distance measure and aggregation method.

One particular usage of clustering regards automatic computing of the number of clusters for colour image segmentation [21]. This approach uses fuzzy c-means algorithms for extracting chromaticity features of colours and trains a Neural-Networks with obtained chromatic data of colours. The trained model may be further used on new colour images to predict the number of clusters in colour images.

Among many clustering libraries, we mention LEAC [36]. It is an open-source library with source code publicly available in the GitHub repository. Thus, once the experiments are also performed on publicly available datasets, the results may be reproducible and also used in other setups for further improvements. Inclusion of 23 state-of-the-art Evolutionary Algorithms for partial clustering within a library that allows easy and fast development and integration of new clustering algorithm represents the solution that we also target when improving the initial version of the MTree clustering algorithm within Weka package.

Another usage of metric trees has been reported recently in [12]. The task is to quickly and efficiently scale-up the problem of shadow rendering for 70 million objects (i.e., triangles) in real-time. The proposed metric tree uses as splitting policy binary space partitioning (BSP) and ternary object partitioning (TOP) for grouping triangles into clusters as precomputed bounding capsules (line-swept spheres).

Finally, the whole clustering process needs validation, and many quality metrics may accomplish this for a wide range of algorithms [18]. Depending on the structure of the dataset, various clustering quality frameworks [23], [38] have been proposed. The key issue that always arises regards choosing the proper similarity and quality metrics [38].

3 Proposed approach

The proposed MTree parametrised clusterer has been firstly proposed in [31] in an attempt to define a new clustering algorithm that has as main business logic the metric trees initially presented in [7] and later in [40] and [43]. The initial C++ implementation approach was designed to cluster students who were defined by three of their obtained grades during one semester. The main shortcoming of the proposed structure was that it as designed only for managing student objects and had several hard-coded parameters needed for building the tree. The most important one is *nrKeys*, which represents the maximum number of students contained in a node (i.e., a cluster). This approach has a critical limitation in the fact that the *splitNote()* method was called based only when a note was full. Other limitations regard the lack of parametrised division policy, distance metrics or other features needed for flexibly running a clustering process.

Later, the initially proposed MTree clustering algorithm has been contributed as an official Weka package [30]. This newly Java-based approach had the goal to be functionally available under Weka workbench as any other clusterer, such that it may be further used in various practical situations. The MTree clusterer from Weka has been used in [37] in a comparative analysis on publicly available images. The results of MTree were inferior such that the limitations were addressed in [9]. As critical improvements, the MTree version from [9] uses off-line dataset preprocessing for finding the optimal number of clusters and adjusts the business logic of the clusterer in terms of division policy and distance metric between instances.

The current proposed version of the MTree cluster represents a flexible parametrised version of the former one in terms of division policy, used distance, the method for finding and setting the number of clusters.

3.1 Definitions and context

The metric space $M = (D,d)$ on a data domain D with the distance function $d : D \times D \to \mathbb{R}$ postulates:

$$Non\ \ negativity : \forall x,y \in D, d(x,y) \geq 0 \tag{1a}$$

$$Symmetry : \forall x,y \in D, d(x,y) = d(y,x) \tag{1b}$$

$$Identity : \forall x,y \in D, x = y \Leftrightarrow d(y,x) = 0 \tag{1c}$$

$$Triangle\ \ inequality : \forall x,y,z \in D, d(x,z) \leq d(x,y) + d(y,z) \tag{1d}$$

The conditions specified above are satisfied by most discrete or continuous distance (or similarity) metrics (or measures): Euclidean, Minkowski, Manhattan, quadratic form distance (i.e., colour histograms, weighted Euclidean distance), edit distance, Jaccard's coefficient or Hausdorff distance. Building clusters of objects in metric spaces relies on the partitioning method and shape of the decision boundary that lays between two adjacent clusters. The partitioning method regards how a set of objects is split into two or more clusters taking into account specific optimisation criteria such as the sum of squared errors (SSE) in case of simple k-means algorithm. Regarding the decision boundary, the two most common options are ball partitioning as in metric spaces and hyperplane partitioning.

As current implementation of the MTree algorithm represents a two-level ball decomposition generalisation of the approach from [40]. The limitations from [40] regard choosing an arbitrary object as the pivot, using only binary splits around the median object, which implies previously sorting the objects and multilayered approach due to recursive construction. From the practical perspective of running a clustering process, these are substantial limitations because ordering may not always be possible, binary splitting may not be useful when dataset consists of many clusters and the notion of the cluster become unclear in a multilayered approach.

Definition 1. The newly designed MTree data structure is a two-level perfectly balanced multiway tree that indexes a set of objects into its leaves which reside only on the second level. After building the tree, the root contains the set of centroids and their corresponding covered radius. On the second level, the leaves represent the clusters containing objects whose distance is smaller or equal to the radius assigned of the corresponding centroid within the root.

The key features for the parametrised MTree implementation are:

1. The possibility to process various object data types provided as input (i.e., image, document, etc.) that is represented as a multidimensional vector.

2. The possibility to set up a particular distance function between objects by direct usage of distance functions that are already available in Weka or by using a newly defined custom function.

3. The possibility to set up a particular division policy that will be used internally used as needed. Practically, the logic of the division policy is managed by the clustering algorithm. This feature needs to be accompanied by a parameter that controls the number of clusters into which full leaf may be splitted. Available options are binary object partitioning (BOP), ternary object partitioning (TOP) or multi-object partitioning (POS).

4. The possibility to set up a specific number of clusters in which the entire dataset will be partitioned, if the number of clusters is known. If the number of clusters is not known, the MTree will find the optimal number of clusters considering the other parameters that have been set.

5. The possibility to compute at request various clustering quality metrics that will give a general idea on the quality of the clustering process. This feature is critical in benchmarking the MTree clustering results against results produced by other clustering algorithms.

The MTree uses only one node data structure for the roots and leaves. In the root node, the instances are represented by centroids, and each element from the *radix* vector represents the covered radius for the corresponding centroid. In the same way, each element from the route vector is an address of the corresponding leaf node. Their vector position accomplishes the correspondence between centroids from the root node and covered radius and leaves. For example, the first element of the instances vector of the root represents the first centroid with a radius defined in the first element of the *radix* vector. The first element of the route vector represents the address of the leaf node that contains objects whose distance to the first centroid is smaller or equal to the covered radius. In the case of leaf nodes, the instances represent the data objects themselves. The *isLeaf* flag is set by the business logic to value *TRUE* such that radix and route vectors are set to null.

Table 1: The structure of MTree node

Field name	Description
nrKeys	The number of objects actually stored in the node.
isLeaf	This flag represents the node type: root or leaf.
radixes	The vector of distances covered by a centroid.
routes	The vector of node addresses to leaf nodes.
instances	The objects from the node (parent or leaf).
parent	The address of the parent node.

Before starting any computation needed for inserting a new object in the MTree, the algorithm checks if we are in a leaf and if the leaf has objects in it. If any of these conditions do not hold than insertion may not take place because insertion may be performed only in a leaf and further splitting may be taken into consideration only if the leaf has objects in it. Further, the leaf is evaluated for splitting parametrised by *evaluatorOfK*, which is a function that determines the optimal number of clusters. This helper function takes as parameter the objects from *treeNode* and outputs *splitEval* as an evaluation of the splitting. If this also is valid, then the leaf node is split into the optimal number of clusters by using a parametrised *divisionPolicy*. The *insertNonFull* function is called to append a new object in the leaf when no splitting is necessary.

The two main ingredients of *mTreeInsert* function are the evaluation of the optimal number of clusters from a leaf and the division policy used for splitting. The current implementation uses a function called *voteK* that computes the optimal number of clusters in a similar way as *NbClust* [6] package in R. The main difference is that *NbClust* uses 30 indices while *voteK* currently integrates only 8 indices: Davies-Bouldin, Dunn, Xi-Beni, Banfeld-Raftery, McClain-Rao, Ray-Turi, Calinski-Harabasz and PBM indices. The architectural design of the package allows the easy call of any method that given an input dataset can compute the optimal number of clusters.

The *nrKeys* represents the number of items (i.e., centroids or items) contained in a node (i.e., root or leaf). For the root node, the items are centroids, and for the leaf node, the items are the sample points. Depending upon the implementation, the centroids may be items from the dataset or computed instances. The *isLeaf* field from the data structure represents a clarification regarding what represent the items from a node: centroids or instances.

As far as our M-tree is concerned, we pay extra attention to the nodes because it is essential whether they are leaves (i.e., terminal nodes containing instances) or internal nodes (i.e., having only centroids) information is critical for the algorithm design.

The instances vector contains the centroid points or items, depending upon the node is either root or leaf. If the node is the root, vector radixes contain the distance covered by each centroid, while the routes vector contains the leaves' addresses. On the other hand, if the node is a leaf, then the field parent includes the root address while the radixes and routes vectors are empty.

The second key ingredient of the *mTreeInsert* function is the division policy. The default option for this parameter is the simple k-means algorithm, but any other strategy may be called as needed. Other options, besides calling particular clustering algorithms as a multi-object partitioning (MOP) strategy, is using binary space partitioning (BSP) or ternary object partitioning (TOP) by simply setting proper parametrised values. Suppose there is no need for node splitting, then insertion reduces to appending the new object into that leaf.

In that case, insertion reduces to appending the new object into that leaf, which is accomplished by calling *insertNonFull* function. Intuitively, when a leaf is not full, we insert a new object; otherwise, we split the leaf. The most crucial difference from former versions or other approaches is that splitting is not called when the number of stored objects reaches a specific value, but when current objects exhibit the property that they are properly clustered, and therefore a split is compulsory.

Algorithm 1 summarizes the business logic for inserting a new object into an existing MTree leaf node.

Algorithm 1 summarises the business logic for inserting a new object into an existing MTree leaf node. The second critical procedure is the one that performs the split of a leaf node as specified in the insertion algorithm. The main ingredient is represented by the *clusteringEvaluator* setup, which allows breaking the leaf into clusters according to with specific division policy and a predetermined optimal number of clusters. The newly obtained clusters are added into a *splitedClusters* vector and returned as output. The main improvement in the split procedure is avoiding splitting a leaf into a hard-coded number of clusters by using a pre-determined optimal number of clusters and parametrising for the division policy for running the effective split. Algorithm 2 summarises the business logic for splitting. The input consists of a tree node (i.e., a cluster) and an evaluator, and the returned value consists of a vector of clusters, called splittedClusters. The evaluator has the task of deciding if the node should remain as a single cluster or be split in two or more clusters. If the evaluator considers more than one cluster in the note, the node will split, and the tree will change its structure.

The bounding volumes of the MTree leaves are circles if objects are 2-dimensional or spheres if objects are 3-dimensional and Euclidean distance is being used. For

Algorithm 1 mTreeInsert (MTree treeNode, Instance newObject)

```
1: if treeNode is leaf and has objects then
2:    Set clusteringEvaluator = getOptimalNrOfClusters (treeNode, evaluatorOfK);
3: end if
4: if splitEval is valid then
5:    nrOfClusters = clusterEval.getNumberOfClusters ();
6:    if nrOfClusters > 1 then
7:       splitNode(treeNode, clusteringEvaluator);
8:    else
9:       insertNonFull(treeNode, newObject);
10:   end if
11: end if
```

Algorithm 2 mTreeSplitNode (MTree treeNode, ClusterEvaluation clusteringEvaluator)

```
1: clusters = getClusters (treeNode, clusteringEvaluator);
2: if size of clusters > 0 then
3:    for each cluster in clusters do
4:       centroidInstance = chooseCenter(cluster);
5:       splitedCluster.addCentroid(centroidInstance.getCentroid());
6:       splitedCluster.setRadix(centroidInstance.getRadix());
7:       splitedClusters.add(splitedCluster);
8:    end for
9: end if
10: Return splitedClusters;
```

n-dimensional spaces, the bounding volumes are hyperspheres, a generalisation for a set of points equally distanced to a given point. This generalisation is valid only for Euclidean distance, as for other distances, the bounding volume is the representative of a sphere for that particular vector space.

3.2 MTree algorithms description

The MTree implementation is aimed for usage by client code in practical scenarios. Although the primary usage scenario addresses the situation for which we know the actual number of clusters, the clusterer may also be used in the case when the data analyst specifies a particular number of clusters depending on the tackled task or in the situation when the number of clusters is not known or does not exist.

In any of the situations mentioned above, the parameters that need to be set are the input data-set, the number of clusters, the distance metric, the evaluator of the number of clusters and the division policy algorithm. The main practical goal of the current version of the MTree clusterer is to verify that it correctly loads the data given specific parameters and creates a data model that validates the ground-truth model from two publicly available data sets. Finally, the clustering validation metrics are verified against other clustering algorithms implemented in Weka workbench.

One key aspect for correctly building the MTree from data regards the order in which the data objects are provided as input. As a general rule, any clustering algorithm is highly sensitive to the order in which data objects are considered in the clustering process. The seed mechanism is the general solution to clustering algorithms and is also used as a standard option in the MTree. For our case, the seed mechanism represents a random shuffle of the order in which the instances are added to the MTree.

The pseudocode of mTreeInsert function presents the structure and logic of operations when inserting a new instance into the tree. The function's parameters are the address of the tree and the item that should be inserted. Since inserting takes place only in leaf nodes, the algorithm first checks that we are in a leaf node and then determines the optimal number of clusters from that leaf. If more than one cluster is found in the leaf, the node is split, and the instance is placed into the appropriate cluster.

The pseudocode of mTreeSplitNode function actually performs the split of a leaf. Along the treeNode, an evaluator is given as parameter, which gathers the parameters needed to determine the clusters. Once the clusters are determined, the centroids and corresponding radixes are placed in the root, the addresses of the new clusters are placed in the routes and instances themselves are placed in the appropriate clusters.

A particular situation occurs when we do not know the correct number of clusters. In this case, the adequate solution is to preprocess the data-set to check clusterability and determine the valid number of clusters if such a number exists. The situation in which the data-set has not a clear well-defined structure may be interpreted in two ways: either the data-set has several possible options as the actual number of clusters, or we do not know the correct number of

clusters. In the first situation, a domain knowledge person should consider the specific data analysis task and choose the correct number of clusters that fit the practical problem. More extensive work needs to be done as a preprocessing step in the latter situation.

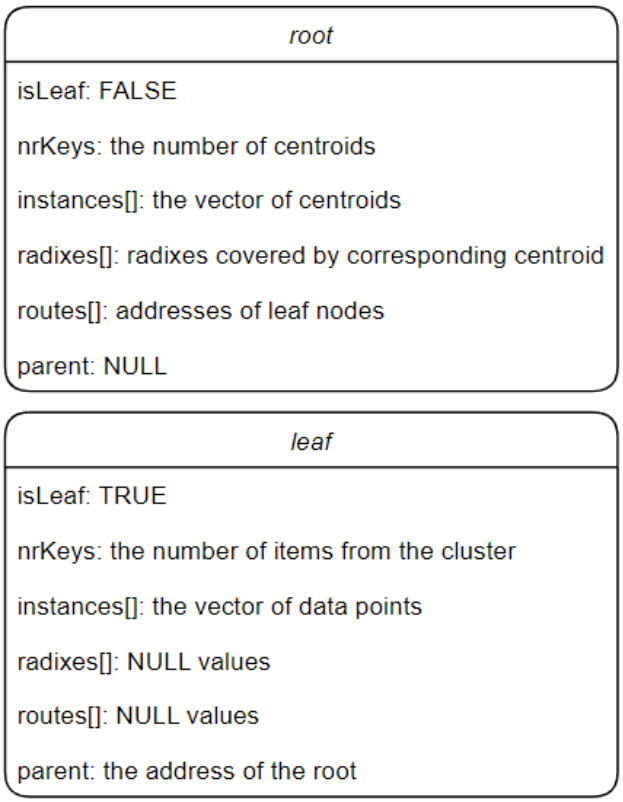


Figure 1: Nodes structure

As a general rule, when the data-set has no structure, the first preprocessing steps should define the number of clusters as a task-dependent value. Then, the centroids should also be defined as representatives for each cluster considered by the domain knowledge person. Finally, the objects from the data-set are added to the pertaining cluster only if a maximal threshold distance from the centroid is not exceeded. The objects that have almost equal distance from centroids are considered outliers and are not added to any cluster. In this way, the data analyst may obtain a clusterable data set with a specific number of clusters. Having a clusterable data-set is a prerequisite for the MTree clustering algorithm and any other algorithm. Therefore, if the data-set does not meet clusterability, building an MTree clusterer or any other clusterer will exhibit undefined behaviour.

Finding the correct number of clusters by using MTree may be performed by running with various parameter settings in terms of the number of clusters and division policy in an attempt to obtain a clusterer whose validation parameters show that the correct patterns have been discovered. In this use case, the MTree data structure is built for finding whatever clusters are to be found.

3.3 Complexity analysis

The complexity analysis of building the MTree from data depends on the number of objects, the number of clusters, the method of finding the optimal number of clusters and the number of distance computations. The number of clusters from the data-set represents the number of splits that need to be performed while building the tree. The most critical operation is finding the optimal number of clusters, which is called after each object insertion. The number of distance computations is related to the number of clusters since distances from the newly inserted object to all the centroids from the root need to be computed to determine the suitable leaf where the insertion should occur. The most time-consuming function is the *getOptimalNrOfClusters* function that is called whenever a new object is to be inserted. We have observed that for a reasonably small number of clusters and a large number of objects, that method *getOptimalNrOfClusters* is used to trigger a split fewer times than the number of clusters. For example, once the number of leaves from MTree has reached the true number of clusters, then looking for the optimal number of clusters becomes useless. Further insertions will be performed in constant time just by determining the proper leaf where the new object needs to be inserted. As stated in [15] the performance of building an MTree with n objects is analogous to that of k-d trees, that is $O(n \log n)$ for worst-case scenario. Depending on the split method the time may increase to $O(n \log^2 n)$ or $O(kn \log n)$ for k dimensions. Still, the currently proposed method is highly sensitive to the order in which objects are being inserted, the seed selection and the particular parameters setup as well as all other clustering algorithms.

The critical property of the MTree is that after correctly building the clusters, the operations of inserting, removing and querying may be performed in $O(\log n)$ time. These aspects are not tested by the current works and need to be further experimentally investigated in practical clustering tasks.

4 Experimental results

4.1 Data-sets description

Experimental results have been performed on two synthetic publicly available data sets from the clustering basic benchmark [19]: *Unbalance* [35] and *Dim2* [28]. *Unbalance* is a synthetic 2-d data-set with 6500 points and 8 Gaussian clusters for which ground truth centroids and partitions are known. Dim2 is also a synthetic 2-d data-set with 1351 points and 9 Gaussian clusters.. Figures 4 and 5 present a plot of the raw input data.

We have chosen two synthetic datasets for which the ground truth centroids and partitions are known because they are suitable for comparing clustering results obtained by MTree algorithm against other ones implemented in Weka workbench.

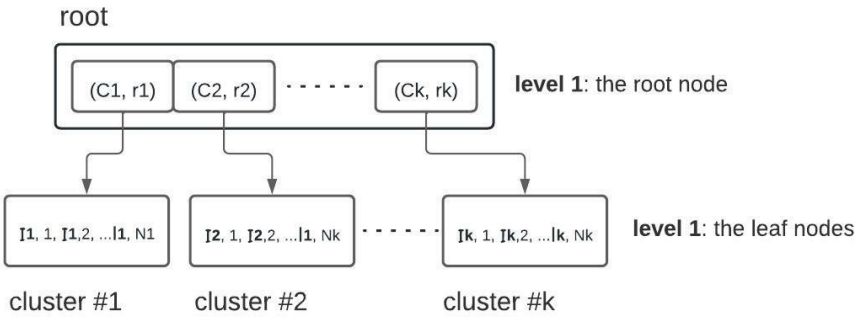


Figure 2: Sample MTree

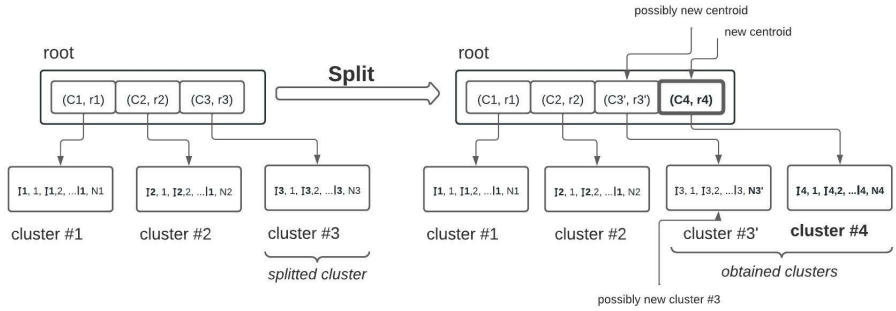


Figure 3: Sample MTree split node

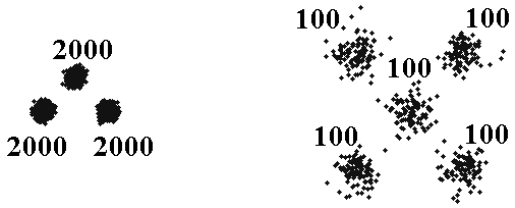


Figure 4: Unbalance dataset

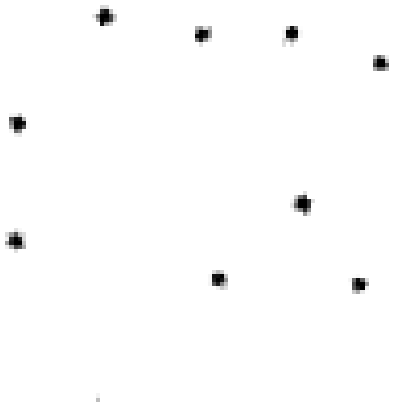


Figure 5: Dim2 dataset

Finally, we have tested the MTree on *Wine* and *Iris* classical datasets from UCI Machine Learning Repository [3]. These real-world datasets were chosen because they may be successfully used in clustering tasks as they are labelled, and classical unsupervised training may correctly determine the classes.

4.2 MTree package description

MTree is implemented in Java and is aimed to be executed within Weka 3.8 workbench by using the *Package manager* tool. The MTree package is based on three classes *Node*, *MTreeBean* and *MTree* along with other three helper classes. Figure 6 shows the software architecture the MTree package as an UML class diagram .

The class *Node* represents a blueprint for a cluster of instances and the *MTreeBean* class contains the root of the MTree. The most important class is *MTree*, which extends *RandomizableClusterer*, which is an abstract class whose direct subclasses are Canopy, Cobweb, FarthestFirst and SimpleKMeans. Further, by implementing the *NumberOfClustersRequestable* and other interfaces, the MTree gets the possibility to be parametrised similarly as other clustering algorithms are in Weka.

The main goal was to obtain a clusterer that may be parameterisable in the same way as already existing clusterers based on interfaces that are already defined in Weka but also offering the possibility of defining new interfaces specific for parameters needed by MTree algorithm. In this way, the

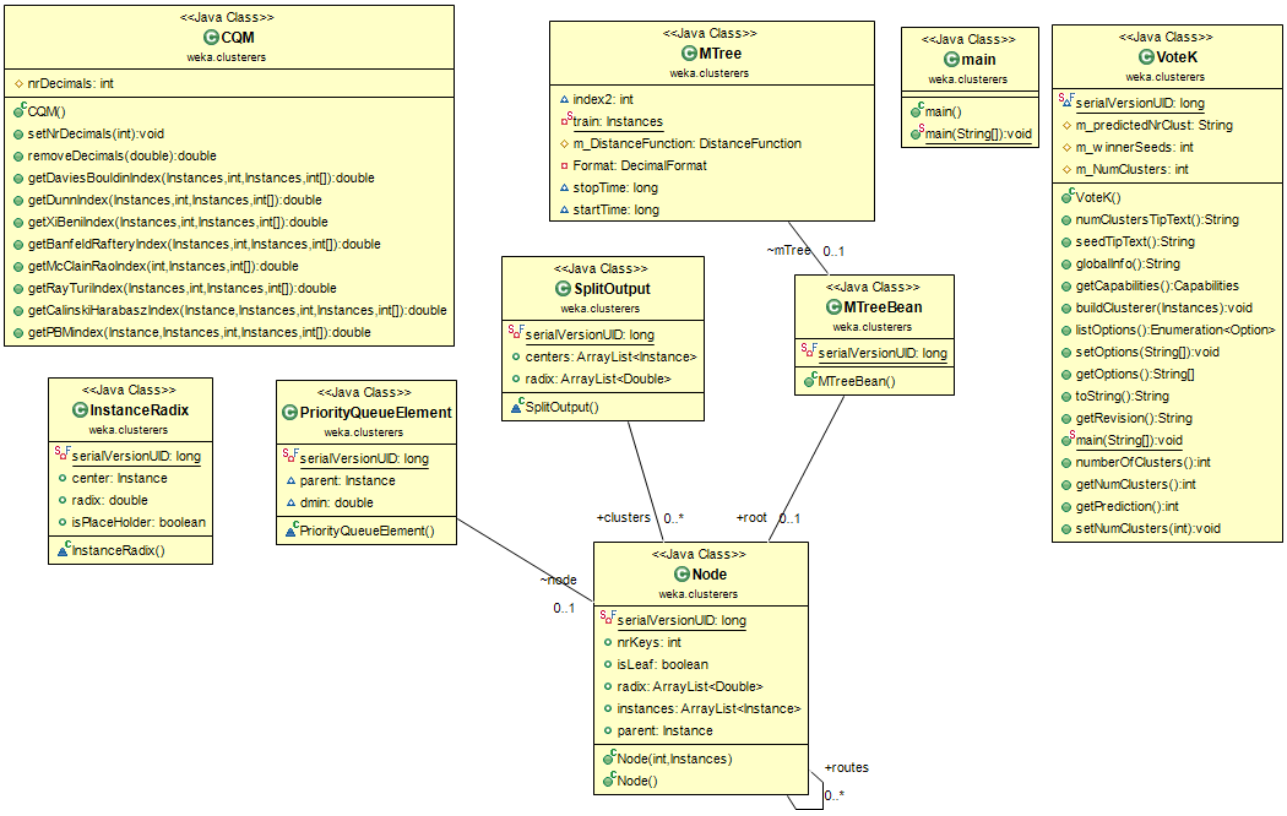


Figure 6: UML class diagram for MTree package

newly obtained MTree can be easily parametrised in the usage of the command-line interface or Weka GUI interface.

4.3 Sample MTree usage

The MTree can be used in three ways. The current implementation provides flexibility for running full experimental runs and benchmarking the performance of an MTree parameter configuration against already existing Weka clustering algorithms.

– *Basic mode, through command line*. This mode allows executing the MTree on any machine that has JVM 1.8 and *weka.jar* version 3.8.3. Figure **??** presents a sample command line execution of the MTree algorithm. This approach is commonly used when batch execution is needed only once for building the clusters and serialising the obtained model (i.e., the distribution of objects into clusters) to persistent storage (i.e., csv, xml, etc.) for later usage is rather tricky.

  The available options when running MTree in command line interface are further presented. ***N*** represents the number of clusters. If we want MTree to decide for itself the number of clusters this option must be set to value -1. ***init*** option may be used for setting the initialization method. ***I*** option is for setting the maximum number of iterations, ***O*** for preserving the order of instances, ***S*** for setting the number of seeds, ***d*** for setting the distance metric, ***findN*** for setting the method for finding the optimal number of clusters and ***splitPolicy*** for setting the method used as splitting policy. Current implementation may use as splitting policy Canopy, Simple k-means, CobWeb, FarthestFirst or HierarchicalClusterer clusterers from Weka.

– *Using the Weka GUI*. This mode is the most user-friendly as the MTree may be used from Weka GUI as any other clustering algorithm. As the MTree package is in the list of official packages, it needs to be installed before usage. Installing the MTree package in Weka is a straightforward procedure as the *MTree.zip* archive is publicly available in SourceForge [30] and the link to the package is available in the list of official packages within the Weka package manager tool.

– *Programmatic way*. The most versatile usage of the MTree is programmatically. This approach allows setting up the parameters at runtime as well as having a ready-to-use in memory MTree object that is ready for querying. This approach allows the usage of the cluster as business logic on a server side in complex applications where client code is performing various queries. Sample code for building the MTree from data is publicly available in [8].

```
java weka.clusterers.MTree -t unbalance.arff \
- N 8 -init 1 -I 1000 -O -S 100 -d 1 -findN weka.clusterers.voteK -splitPolicy 0
```

Figure 7: Command line execution of MTree

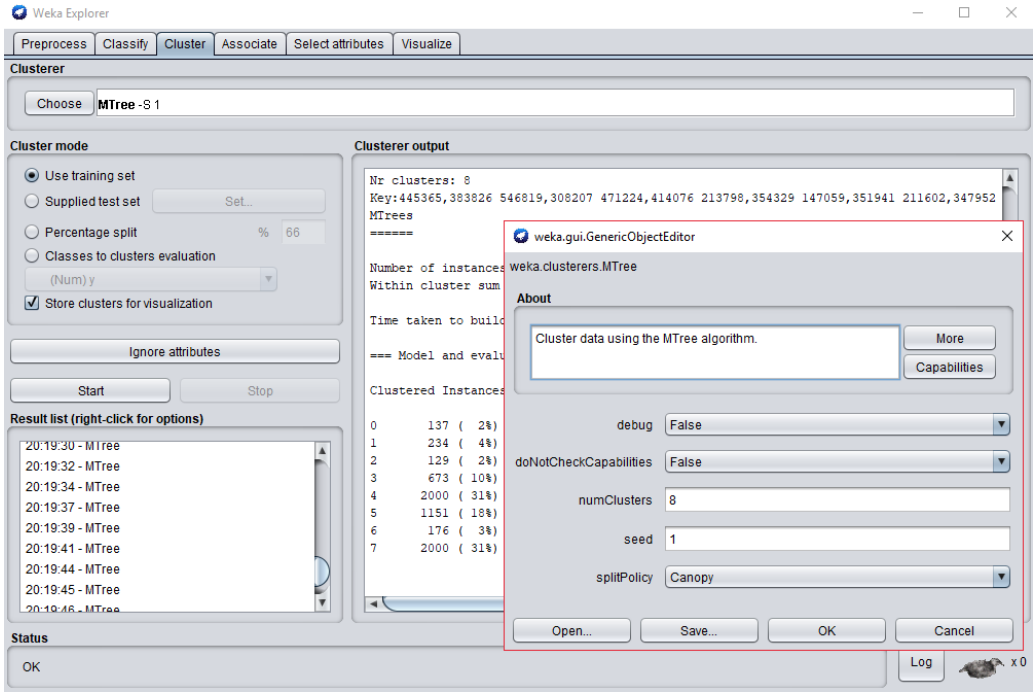


Figure 8: Execution of MTree in Weka GUI

4.4 Sample runs on Unbalance and Dim2 synthetic data-sets

The newly released parametrised MTree implementation has been tested against two publicly available synthetic data sets: *Unbalance* and *Dim2*. The client code that calls the MTree package is publicly available at [8], as further presented experimental results were obtained by programmatically running the MTree implementation.

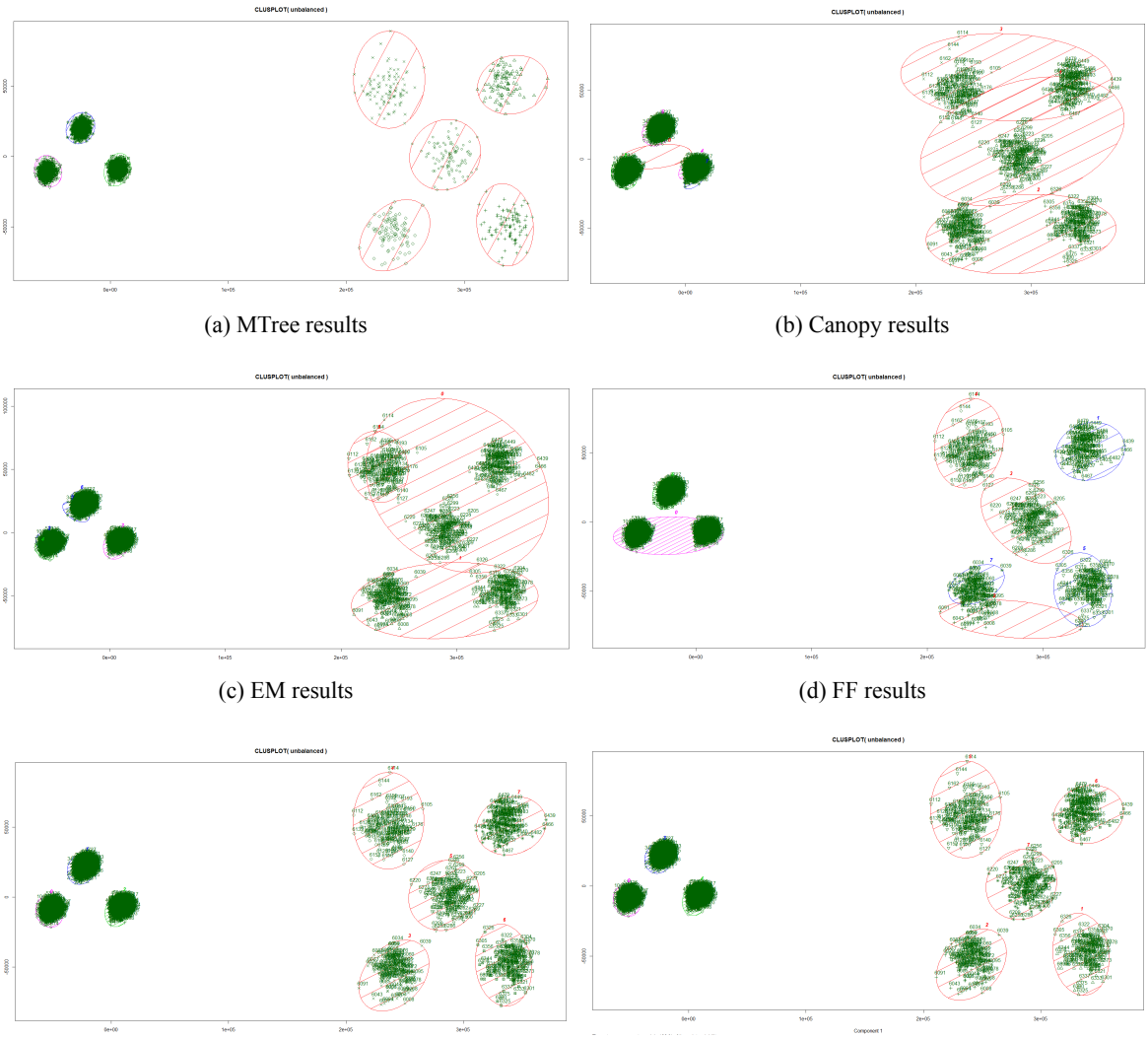
Figures 9 and 10 present the experimental results of running the MTree clusters along other five clusterers implemented in Weka workbench: Canopy, EM (expectation-maximization), FF (Farthest First), HC (Hierarchical Clustering) and SKM (Simple KMeans).

As figure 9 clearly shows, the MTree correctly determines the clusters by using 100 seeds and Canopy for split policy. As a general rule, the clustering result exhibits undefined behaviour regarding the number of seeds, such that correct results may be obtained sometimes for only 10 seeds and sometimes for 1000 seeds. Here is a summary of the experimental results of the other five algorithms:

- *Canopy* algorithm fails to determine the correct number of clusters and the found distribution into clusters is wrong.
- *EM* algorithm fails to determine the correct number of clusters although in many situations it is used for this task as it does not require the value of K. The obtained clusters are reasonable fine with two exceptions: cluster 0 puts together three real clusters and cluster 1 puts together two real clusters.
- *FF* algorithm correctly determines the number of clusters but misses to determine two of them correctly: cluster 0 puts together two real clusters and cluster 2 is composed of objects belonging to two distinct clusters.
- *HC* algorithm correctly solves the task.
- *SKM* algorithm correctly solves the task after finetuning the parameters: 100 seeds, usage of kmeans++ [2] for seed optimisation and maximum 10,000 iterations.

As figure 10 clearly shows, the MTree correctly determines the clusters on a more difficult task by using only 10 seeds. The other investigated algorithms provide the following results:

- *Canopy* algorithm fails to determine the correct number of clusters and the found distribution into clusters is wrong, as it puts together in a cluster objects from two clusters.
- *EM* algorithm correctly determines the clusters.
- *FF* algorithm fails to determine the correct number of clusters and misses to determine one of them correctly.



(a) MTree results (b) Canopy results

(c) EM results (d) FF results

(e) HC results (f) SKM results

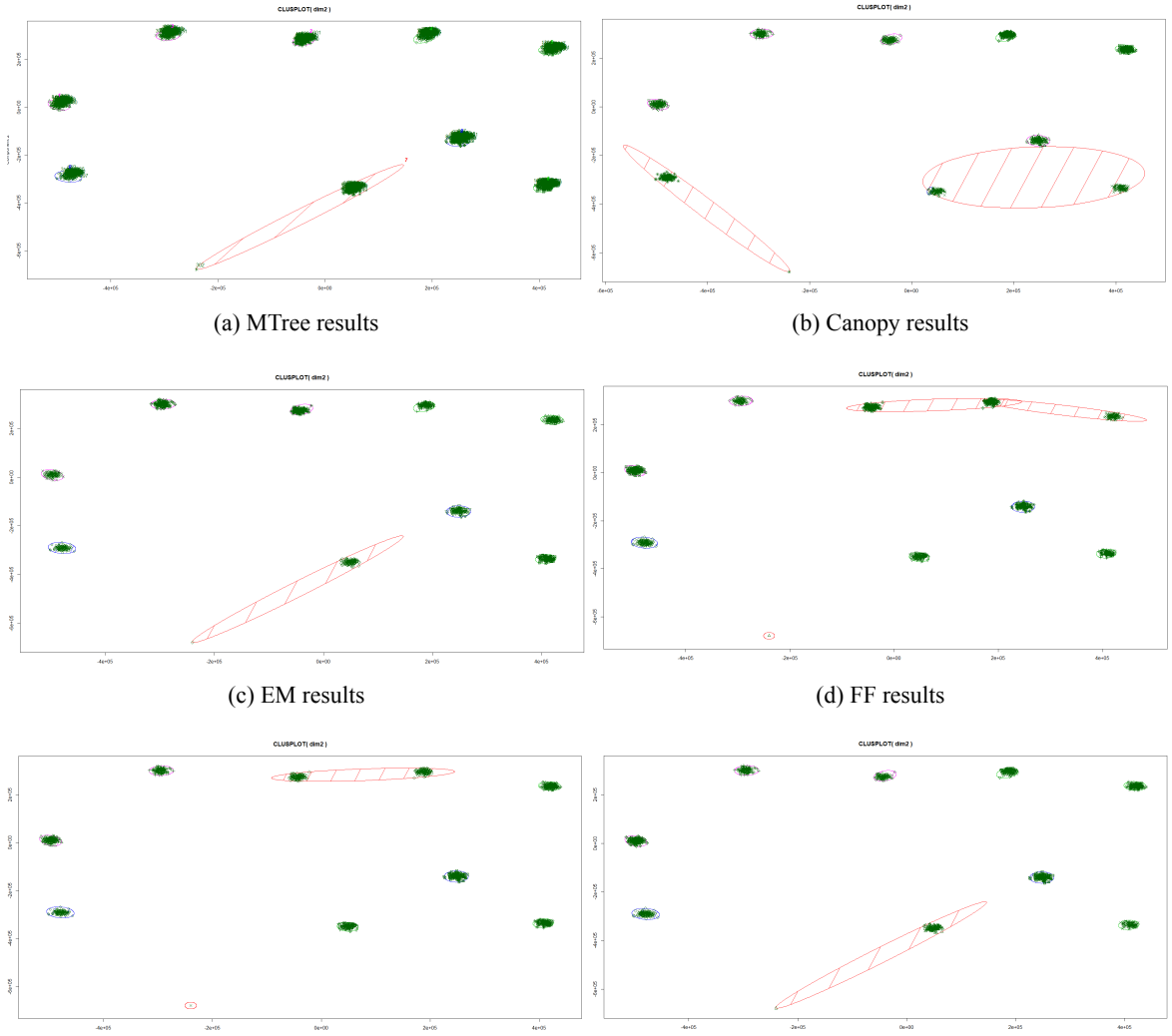
Figure 9: Clustering results on *Unbalance* dataset

Table 2: Running times statistics (measured in seconds)

Algorithm	**Unbalance dataset**	**Dim2 dataset**
MTree	0.3 [per seed]	0.06 [per seed]
Canopy	0.01	0.1
EM	8.21 [per tuned seed]	0.55 [per tuned seed]
FF	0.01 [per seed]	0.01 [per seed]
HC	605.42	4.11
SKM	0.06 [per seed]	0.01 [per seed]

Table 3: Performance results on real world datasets

Algorithm	**Accuracy Wine**	**Accuracy Iris**
MTree+Canopy	0.94382	0.89261
MTree+cKMs	0.92134	0.89261
MTree+FF	0.93258	0.88590
MTree+HC	0.89887	0.89261
KMeans	0.93258	0.88590



(a) MTree results (b) Canopy results

(c) EM results (d) FF results

(e) HC results (f) SKM results

Figure 10: Clustering results on *Dim2* dataset

The result shows that objects from one real cluster are split between two real clusters.

– *HC* fails to determine the correct number of clusters and reports one found cluster as a join between two real clusters.

– *SKM* algorithm correctly solves the task after fine tuning the parameters: 10 seeds, usage of kmeans++ [2] for seed optimisation and maximum 10,000 iterations.

Experimental results show that the first K instances have the most significant impact over the final result, where K is the number of clusters. Thus, the current implementation uses kmeans++ [2] seed optimisation, so the first K instances that are added to the MTree are a rather sparse one from another.

Table 2 summarizes the running time statistics for all six algorithms on both data-sets. In the case of EM algorithm, each tuned seed has been obtained by more iterations, and more k-means runs, a fact that explains the more significant running time. HC and Canopy do not have seeds and Canopy's poor results on both data-sets were obtained regardless of the configuration parameters. The number of seeds for the algorithms that correctly solved the *dim2* data-set has been set to 10.

Finally, the SSE (sum of squared error), as well as the assignment of objects, are correctly computed for MTree as they compute to the same values as SKM of 4.2471 and 0.3367 for *unbalance* and *dim2* datasets, respectively. Therefore, the clusters produced by the MTree are valid and represent the real ones from the datasets and SSE represents a good optimisation metric for these datasets.

4.5 Sample runs on Iris and Wine real-world data-sets

Wine data was normalised and then MTree was used on all 13 features. The algorithm was run on 100 seeds and the best result was targeting to be with the best (smallest) SSE. As can be seen from the table, smaller SSE does not always provide the best accuracy, with a SSE value of 66 against 68 but an accuracy of 89 against 94.This suggests that a different cluster quality metrics may be able to improve the performance of the proposed algorithm. KMeans run on 100 seeds obtains 93% accuracy or 12 wrong predictions.

Iris data was normalised and MTree used on all 4 features. As on the previous data-set, the algorithm was run on 100 seeds and best SSE was targeted. It is interesting to notice that different SSE provide the same accuracy, it seems that the algorithm converged with 16 wrong predictions being its best. MTree+cobweb is not able to cluster the data. On the same data, KMeans run on 100 seeds obtains 88% accuracy or 17 wrong predictions.

Table 3 presents accuracy results of MTree parametrised by various splitting algorithms (i.e., Canopy, KMeans, Farthest First and Hierarchical clustering) against baseline KMeans algorithm. Experimental results show the MTree clustering algorithm correctly finds groups at least as good as simple KMeans algorithm.

5 Conclusions and future work

This paper presents an improved parametrised MTree clusters for Weka workbench. The experimental results show that MTree correctly solves two synthetic datasets for which the correct structure (i.e., number of clusters, centroids and distribution) is known. More, five other clustering algorithms implemented in Weka are outperformed in various situations due to that fact that they do not solve the clustering task correctly or need intensive tuning for solving it.

Still, the current approach is used only for *sanity check* of the clustering capabilities of the MTree implementation, rather than solving a particular clustering task on a real dataset. The implementation is Java-based and is available as open source Weka package. The experimental results are correct and promising such that further development under Weka offers the possibility of proper benchmarking of further clustering tasks that may be taken into consideration.

The main contributions are summarised as follows:

1. An updated parametrised version of the MTree package is presented. The parametrisation mainly regards used distance metric, the method for finding and setting the number of clusters and the division policy. The data structure can load various object types after being properly processed as well as providing validation insights.

2. The proposed software architecture of the MTree enables parameterisation through easy integration of other internal algorithmic strategies that perform key tasks within the business logic.

3. The implementation of the MTreeis available as an open source package in Weka workbench. This approach gives the opportunity for further usage and benchmarking against other clustering algorithms.

4. The experiments demonstrate that the proposed approach outperforms current clustering algorithms on two datasets.

Future works may take into consideration extending the *voteK* algorithm as a Java implementation of the already existing R package NbClust. Extending *voteK* should take into consideration the available clustering quality indices and parametrisation capabilities. In terms of internal business logic, MTree may try different approaches regarding the order in which objects are processed when building the tree. One option is to cluster the outlier objects later in the process.

As the most expensive operations are finding the optimal number of clusters and splitting, one option is trying to predict how the insertion of an object will impact the tree in

terms of triggering a split. Checking for the optimal number of clusters should be performed only when an insert is highly to determine a split, as most inserts do not require a split, especially when the dataset has a well-defined structure.

MTree currently implements range query and kNN query. These implementations should be further tested in practical real data scenarios. Other tasks in which MTree may be also used are outlier detection and finding the correct number of clusters in a dataset. Finally, MTree algorithm may be further tested for finding patterns in data in the situation when internal structure is not known.

Acknowledgement

This work was partially supported by the grant 135C/2021 ”Development of software applications that integrate machine learning algorithms”, financed by the University of Craiova.

References

[1] Andreas Adolfsson, Margareta Ackerman, and Naomi C Brownstein. “To cluster, or not to cluster: An analysis of clusterability methods”. In: *Pattern Recognition* 88 (2019), pp. 13–26. DOI: 10.1016/j.patcog.2018.10.026.

[2] David Arthur and Sergei Vassilvitskii. “k-means++: The advantages of careful seeding”. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.

[3] Catherine L Blake and Christopher J Merz. *UCI repository of machine learning databases, 1998*. 1998.

[4] Roberto Caldelli et al. “Fast image clustering of unknown source images”. In: Jan. 2011, pp. 1–5. DOI: 10.1109/WIFS.2010.5711454.

[5] Jianlong Chang et al. “Deep adaptive image clustering”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 5879–5887. DOI: 10.1109/iccv.2017.626.

[6] Malika Charrad et al. “NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set”. In: *Journal of Statistical Software* 61 (Oct. 2014), pp. 1–36. DOI: 10.18637/jss.v061.i06.

[7] Paolo Ciaccia et al. “Indexing metric spaces with m-tree.” In: *SEBD*. Vol. 97. 1997, pp. 67–86.

[8] Marius Andrei Ciurez. *MTree client code*. https://github.com/kyko007/Cordoba/tree/master/MTree. 2019.

[9] Marius Andrei Ciurez and Marian Cristian Mihaescu. “Improved Architectural Redesign of MTree Clusterer in the Context of Image Segmentation”. In: *International Conference on Intelligent Data Engineering and Automated Learning*. Springer. 2018, pp. 99–106. DOI: 10.29007/sm6x.

[10] Abhisek Dash et al. “Image Clustering without Ground Truth”. In: *CoRR* (Oct. 2016). DOI: 10.48550/arXiv.1610.07758.

[11] Jiangzhou Deng, Junpeng Guo, and Yong Wang. “A Novel K-medoids clustering recommendation algorithm based on probability distribution for collaborative filtering”. In: *Knowledge-Based Systems* (Mar. 2019). DOI: 10.1016/j.knosys.2019.03.009.

[12] F Deves et al. “Scalable Real-Time Shadows using Clustering and Metric Trees”. In: *Eurographics Symposium on Rendering*. Karlsruhe, Germany, July 2018, pp. 1–12. DOI: 10.2312/sre20181175. URL: https://hal.archives-ouvertes.fr/hal-02089095.

[13] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. “Image segmentation using K-means clustering algorithm and subtractive clustering algorithm”. In: *Procedia Computer Science* 54 (2015), pp. 764–771. DOI: 10.1016/j.procs.2015.06.090.

[14] Gianni A Di Caro, Frederick Ducatelle, and L Gambardella. “A fully distributed communication-based approach for spatial clustering in robotic swarms”. In: *Proceedings of the 2nd Autonomous Robots and Multirobot Systems Workshop (ARMS), affiliated with the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)(Valencia, Spain, June 5)*. Citeseer. 2012, pp. 153–171.

[15] Herbert Edelsbrunner. *Algorithms in combinatorial geometry*. Vol. 10. Springer Science & Business Media, 2012. DOI: 10.1007/978-3-642-61568-9.

[16] Frank Eibe, MA Hall, and IH Witten. “The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques”. In: *Morgan Kaufmann* (2016). DOI: 10.1016/B978-0-12-804291-5.00024-6.

[17] Ahmed Ali Abdalla Esmin, Rodrigo A. Coelho, and Stan Matwin. “A review on particle swarm optimization algorithm and its variants to clustering high-dimensional data”. In: *Artif. Intell. Rev.* 44.1 (2015), pp. 23–45. DOI: 10.1007/s10462-013-9400-4.

[18] Adil Fahad et al. “A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis”. In: *IEEE Trans. Emerging Topics Comput.* 2.3 (2014), pp. 267–279. DOI: 10.1109/tetc.2014.2330519.

[19] Pasi Fränti and Sami Sieranoja. "K-means properties on six clustering benchmark datasets". In: *Applied Intelligence* 48.12 (2018), pp. 4743–4759. DOI: 10.1007/s10489-018-1238-7.

[20] Guojun Gan, Chaoqun Ma, and Jianhong Wu. *Data Clustering: Theory, Algorithms, and Applications*. Society for Industrial and Applied Mathematics, 2007. DOI: 10.1137/1.9780898718348.

[21] Farid Garcia-Lamont et al. "Automatic computing of number of clusters for color image segmentation employing fuzzy c-means by extracting chromaticity features of colors". In: *Pattern Analysis and Applications* 23 (2020), pp. 59–84. DOI: 10.1007/s10044-018-0729-9.

[22] Melvin Gauci et al. "Clustering objects with robots that do not compute". In: *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2014, pp. 421–428.

[23] Ángel Castellanos Gonzáles, Juan Manuel Cigarrán, and Ana García-Serrano. "Formal concept analysis for topic detection: A clustering quality experimental analysis". In: *Information Systems* 66 (2017), pp. 24–42. ISSN: 0306-4379. DOI: 10.1016/j.is.2017.01.008.

[24] Suchita Goswami and Lalit Kumar P Bhaiya. "Brain tumour detection using unsupervised learning based neural network". In: *2013 International Conference on Communication Systems and Network Technologies*. IEEE. 2013, pp. 573–577. DOI: 10.1109/csnt.2013.123.

[25] Sudipto Guha and Nina Mishra. "Clustering Data Streams". In: *Data Stream Management - Processing High-Speed Data Streams*. Ed. by Minos N. Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Springer, 2016, pp. 169–187. DOI: 10.1049/iet-smt.2018.5389.

[26] Givanna H. Putri et al. "ChronoClust: Density-based clustering and cluster tracking in high-dimensional time-series data". In: *Knowledge-Based Systems* 174 (Feb. 2019). DOI: 10.1016/j.knosys.2019.02.018.

[27] K. Anil Jain and Aditya Vailaya. "Image retrieval using color and shape". In: *Pattern Recognition* 29.8 (1996), pp. 1233–1244. DOI: 10.1016/0031-3203(95)00160-3.

[28] Ismo Kärkkäinen and Pasi Fränti. "Gradual model generator for single-pass clustering". In: *Pattern Recognition* 40.3 (2007), pp. 784–795. DOI: 10.1016/j.patcog.2006.06.023.

[29] Manish Maheshwari, Sanjay Silakari, and Mahesh Motwani. "Image clustering using color and texture". In: *Computational Intelligence, Communication Systems and Networks*. IEEE, 2009, pp. 403–408. DOI: 10.1109/CICSYN.2009.69.

[30] Marian Cristian Mihaescu. *MTree Clusterer*. Accessed: 2019-05-30. URL: http://weka.sourceforge.net/packageMetaData/MTreeClusterer/index.html.

[31] Marian Cristian Mihaescu and Dumitru Dan Burdescu. "Using M tree data structure as unsupervised classification method". In: *Informatica* 36.2 (2012).

[32] Anton Milan et al. "Joint tracking and segmentation of multiple targets". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, pp. 5397–5406. DOI: 10.1109/cvpr.2015.7299178.

[33] Jose A Miñarro-Giménez, Markus Kreuzthaler, and Stefan Schulz. "Knowledge Extraction from MEDLINE by Combining Clustering with Natural Language Processing". In: *AMIA Annual Symposium Proceedings*. Vol. 2015. American Medical Informatics Association. 2015, p. 915.

[34] Traian Rebedea and Ştefan Trăuşan-Matu. "Autonomous News Clustering and Classification for an Intelligent Web Portal". In: *Foundations of Intelligent Systems*. Springer Berlin Heidelberg, 2008, pp. 477–486. DOI: 10.1007/978-3-540-68123-6_52.

[35] Mohammad Rezaei and Pasi Fränti. "Set matching measures for external cluster validity". In: *IEEE Transactions on Knowledge and Data Engineering* 28.8 (2016), pp. 2173–2186. DOI: 10.1109/TKDE.2016.2551240.

[36] Hermes Robles et al. "LEAC: An efficient library for clustering with evolutionary algorithms". In: *Knowledge-Based Systems* (May 2019). DOI: 10.1016/j.knosys.2019.05.008.

[37] Érick Oliveira Rodrigues et al. "K-MS: a novel clustering algorithm based on morphological reconstruction". In: *Pattern Recognition* 66 (2017), pp. 392–403. DOI: 10.1016/j.patcog.2016.12.027.

[38] Tiago Rodrigues Lopes dos Santos and Luis E. Zárate. "Categorical data clustering: What similarity measure to recommend?" In: *Expert Syst. Appl.* 42.3 (2015), pp. 1247–1260. DOI: 10.1016/j.eswa.2014.09.012.

[39] Lincoln F Silva et al. "Hybrid analysis for indicating patients with breast cancer using temperature time series". In: *Computer methods and programs in biomedicine* 130 (2016), pp. 142–153. DOI: 10.1016/j.cmpb.2016.03.002.

[40] Jeffrey K Uhlmann. “Satisfying general proximity/similarity queries with metric trees”. In: *Information processing letters* 40.4 (1991), pp. 175–179. DOI: 10.1016/0020-0190(91)90074-R.

[41] Ulrike Von Luxburg, Robert C Williamson, and Isabelle Guyon. “Clustering: Science or art?” In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. 2012, pp. 65–79.

[42] Zhimin Wang et al. “Adaptive spatial information-theoretic clustering for image segmentation”. In: *Pattern Recognition* (Sept. 2009), pp. 2029–2044. DOI: 10.1016/j.patcog.2009.01.023.

[43] Pavel Zezula et al. *Similarity search: the metric space approach*. Vol. 32. Springer Science & Business Media, 2006. DOI: 10.1007/0-387-29151-2.

[44] Shibing Zhou and Zhenyuan Xu. “Automatic grayscale image segmentation based on Affinity Propagation clustering”. In: *Pattern Analysis and Applications* (Feb. 2019). DOI: 10.1007/s10044-019-00785-4.